Bachelor Project

Czech Technical University in Prague

F3

Faculty of Electrical Engineering
Open informatics

Virtual Bumper for Tracked Ground Robot from Depth Data

Vít Zlámal

Supervisor: doc. Tomáš Svoboda, Ph.D.
Field of study: Open informatics
May 2017

Czech Technical University in Prague
Faculty of Electrical Engineering

Department of Cybernetics

BACHELOR PROJECT ASSIGNMENT

Student: Vít Z l á m a l

Study programme: Open Informatics

Specialisation: Computer and Information Science

Title of Bachelor Project: Virtual Bumper for Tracked Ground Robot from Depth Data

Guidelines:

An outdoor tracked robot used in Urban Search and Rescue (USAR) operations must react to possible obstacles in close vicinity regardless whether it is teleoperated or drives autonomously [1]. The robot is equipped with variety of sensors also measuring depth (Lidar, RealSense camera). Standard simple obstacle detections are not applicable as the robot can actually traverse many almost-obstacles [2]. Time-to-collision problem is even more complicated as the robot morphology is changing [2]. Research existing approaches, like [3] and design, implement, integrate (ROS.org) and experimentally verify a virtual bumper algorithm that provides an estimate of time-to-contact for any part of the robot. As a side product the algorithm may provide data for the adaptive traversal algorithm [2]. Design and implement a simple calibration procedure that aligns the RealSense data with the main coordinate system of the robot. The calibration may use a simplified scene like with one dominant horizontal plane.

Bibliography/Sources:

[1] Ivana Kruijff-Korbayová, Francis Colas, Mario Gianni, Fiora Pirri, Joachim de Greeff, Koen V. Hindriks, Mark A. Neerincx, Petter Ögren, Tomás Svoboda, Rainer Worst: TRADR Project: Long-Term Human-Robot Teaming for Robot Assisted Disaster Response. KI 29(2): 193-201 (2015)

[2] Martin Pecka, Karel Zimmermann, Michal Reinstein, Tomás Svoboda: Controlling Robot Morphology From Incomplete Measurements. IEEE Trans. Industrial Electronics 64(2): 1773-1782 (2017)

[3] Jia Pan and Ioan A. S and Sachin Chitta and Dinesh Manocha: Efficient Collision Detection and Distance Computation with Realtime Sensor Data. In IEEE International Conference on Robotics and Automation (ICRA), 2013

Bachelor Project Supervisor: doc. Ing. Tomáš Svoboda, Ph.D.

Valid until: the end of the summer semester of academic year 2017/2018

L.S.

prof. Dr. Ing. Jan Kybic
Head of Department

prof. Ing. Pavel Ripka, CSc.
Dean

Prague, January 12, 2017

Acknowledgements

I would like to thank my supervisor Tomáš Svoboda for his guidance, infinite patience and for giving me the opportunity to be involved in this spectacular project.

I am also grateful to Martin Pecka for helping me with ROS and everything else around project.

At last but not least I would like to thank the team working around the robot and cooperating on this project.

Declaration

I declare that I have developed the presented work independently and that I have listed all information sources used in accordance with the Methodical guidelines on maintaining ethical principles during the preparation of higher education theses.

In Prague, 26. May 2017

. . . . . .

Abstract

Human operator of an outdoor tracked ground robot can overlook obstacles and crash in to them. Those collisions can damage the robot or in some cases whole robot can be lost due to collision when the robot gets stuck in dangerous area.

In this thesis we describe how to build virtual bumper using depth data (point clouds), which were captured by Intel® RealSense™ (RealSense) sensor.

The result of our work is package for Robot Operating System (ROS)[4], which can stop the robot in critical situations before collision. Whole project is implemented in C++ and uses Flexible Collision Library (FCL)[3] for collision checking.

Keywords: FCL, Flexible Collision Library, RealSense, point cloud, ROS, Virtual bumper

Supervisor: doc. Tomáš Svoboda, Ph.D.

Abstrakt

Pásový robot ovládáný operátorem, který však může přehlédnout překážky a nasledně do nich narazit. Takovéto kolize mohou vést k poškození robota, případně i k jeho ztrátě pokud robot zůstane neovladatelný v nebezpečné oblasti.

V této práci popisujeme jak lze vytvořit virtuální nárazník za použití hloubkových dat (point cloudů), který jsme snímli senzorem Intel® RealSense™ (RealSense).

Výsledkem práce je balíček pro Robotický operační systém (ROS)[4], který dokáže v kritické situaci zastavit robota dříve, než dojde ke kolizi. Celý projekt je implementován v C++ a s využitím Flexible Collision library (FCL)[3].

Klíčová slova: FCL, Flexible Collision Library, RealSense, point cloud, ROS, Virtualní nárazník

Překlad názvu: Virtuální nárazník pro pásového robota z hloubkových dat

Contents

Figures

Tables

Chapter 1

Introduction

Human operator of an outdoor tracked ground robot can overlook obstacles and crash in to them. Those collisions can damage the robot or in some cases whole robot can be lost due to collision when the robot gets stuck in dangerous area.

In this thesis we describe how to build virtual bumper using depth data (point clouds), which were captured by Intel® RealSense™ (RealSense) sensor.

The result of our work is package for Robot Operating System (ROS)[4], which can stop the robot in critical situations before collision. Whole project is implemented in C++ and uses Flexible Collision Library (FCL)[3] for collision checking.

1.1 State of the art

Virtual bumpers designed for vehicles provides feedback in form of distance measurement. Bumpers in modern cars provide audio alarm with frequency increasing with decreasing distance of a car from an obstacle. This type of bumper is not suitable for our robot, because urban search and rescue (USAR) robots have different requirements. USAR robots can traverse obstacles that would be considered like dangerous from simple distance measurement. Therefor algorithm using ultrasonic sensors are not a good option for our USAR robot, because shape of an obstacle can not be recognized [14].

Collision checking is essential for virtual bumpers where recognition of the obstacles shape is needed. Plethora of collision checking algorithms has been developed and placed into libraries such as Bullet [7], ODE [8], PQP [9] or V-Collide [10] for wider use.

However, almost all these algorithms were originally made for environment where objects are represented in form of meshes or geometric primitives. These representations are very different from data recorded by Intel® RealSense™ (RealSense) camera in real situations. RealSense captures only part of an environment and the output can be noisy. An example of point cloud is shown on figure 2.1.

Flexible collision library [1] is designed to handle data from depth sensors, such as laser sensors or stereo cameras, and provide sufficient collision check

system. FCL is also fast enough for real time collision checking, therefor it is good choice for building virtual bumper.

1.2 Motivation

Robots used in Long-Term Human-Robot Teaming for Disaster Response (TRADR)[6] project are designed to help rescue teams in USAR operations. Various kinds of robots are collaborating with humans and exploring environment of disaster. This information improves team members understanding how to operate in dangerous area.

Tracked ground robot is equipped with diverse sensors like thermal camera, LIDAR and RealSense. The robot can be operated autonomously or by the operator. When the robot is operated by human member of a team, mistakes in steering are made and that leads to collisions. Sensors can be broken or in some cases whole robot can be lost due to collision when the robot gets stuck in dangerous area.

In effort to protect the robot before collision virtual Bumper have to be developed. Bumper needs to stop the robot in dangerous situation before crash.

1.3 Problem specification

The goal is to create virtual bumper for TRADR robot, that would be able to evade collisions in harsh unstructured indoor and outdoor environments. The robot have to be stopped by virtual bumper in risky situations as soon as is possible for maximum safety. But it is also necessary to minimize false alarm detection in order to not increase cognitive load of the robots operator. Algorithm must adapt to robots ability to traverse obstacles which is changing with robots flippers arrangement. Depending on the flippers configurations some terrain may be recognized as a dangerous obstacle or a difficult but traversable terrain[2]. Example of difficult terrain can be stairs because the robot can traverse them only with right configuration of flippers.

For our purpose we decided to use RealSense sensor because of its speed and ability to produce depth data. However depth data from RealSense are burdened by random noises. In connection with light condition RealSense produce different amount of data. Color of an object also can be factor in density of data produced by RealSense. Virtual bumper has to deal with all these inaccuracies.

Chapter 2

Essential terms

2.1 Point cloud

Point cloud is set of 3D points in a coordinate system. Our robot generates point clouds through RealSense sensor in 3D coordinate system with origins in middle of the sensor. Points in a point cloud can be understood as a sample depth/distance measurements. It is not a complete representation. Only directly visible scene parts are sampled. Density of samples varies depending on the shape of the terrain/obstacles and its distance from the robot.

Figure 2.1: Point cloud of stairs

2.2 Obstacle

First of all we have to define what is danger obstacle that that have to be recognized. Simply everything that can not be traverse is an dangerous obstacle and everything else is not, but there is problem because these two sets have a common intersection. Sometimes exactly the same obstacle can be dangerous and safe depending on robots flippers state. As shown in figure 2.2 stairs are example of difficult obstacle. More common obstacles are walls,

pillars, debris and holes where we can strictly decide what is dangerous and what is not.

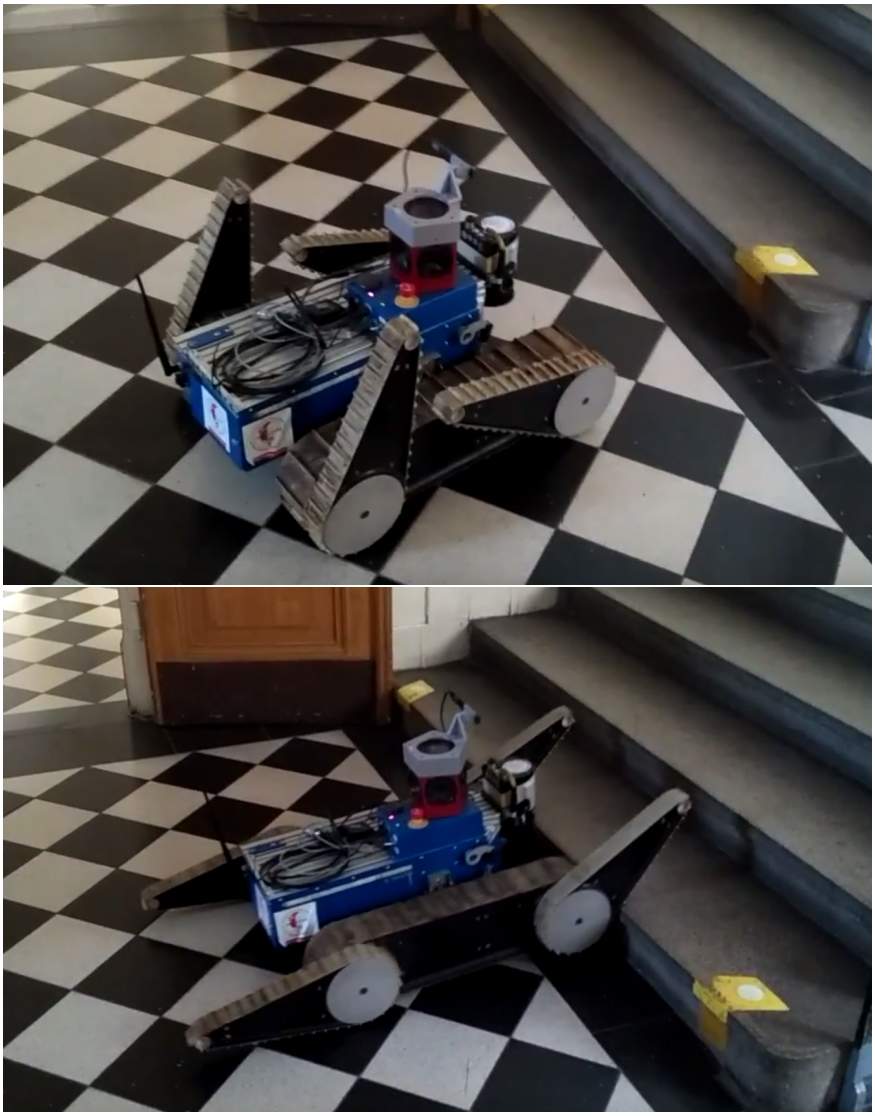


Figure 2.2: Different settings of flippers

2.3 Robot Operating System

The Robot Operating System (ROS) is a flexible framework for writing robot software. It is a collection of tools, libraries, and conventions that aim to simplify the task of creating complex and robust robot behavior across a wide variety of robotic platforms.[4]

The TRADR robots are running under the ROS system. This framework provides elegant way to operate and combine different components of robot. The whole system is build from standardized nodes which makes it very

modular. Nodes can communicate via topics between each other in M to N relation.

All this features provides distributed system where nodes can be implemented in different languages and can run on different machines. Only communication protocol have to be respected. Also it leads us to create Small parts of code that can be integrated together only with functionality we need.

Every part of ROS have to be in catkin package [5]. Every package requires `CMakeFile.txt` and `package.xml` file. In these files are building information including dependencies requirements. This is very useful for example when `rosdep` command downloads all dependencies for specific package.

2.3.1 ROS core

Ros core is set of nodes and programs that are essential for every ROS application. ROScore must be running in order for ROS nodes to communicate.[5]

2.3.2 Nodes

Node is process that make computation. Nodes are combined in to a graph and communicate with one another.

The system that controls our TRADR robot is compose of many nodes where each node have its own purpose (such as: `path_planer`, `laser_proximity_checker`, `virtual_bumper`). This structure have advantages such as fault tolerance, code simplify and independence on programing language. For example `virtual_bumper` node is written in C++ and `terrain_shape_estimation` is implemented in python. Every running node has its unique name which identifies it to rest of the system.

2.3.3 Topics

Topics are named buses with anonymous subscriber and publisher which are used to transfer messages between nodes [5]. This allows nodes to subscribe and publish information. Topics can be subscribed by multiple nodes and nodes can subscribe multiple topics which makes M to N relation.

2.3.4 Ros spin

Ros spin is endless loop that let nodes to stay alive. When callback method finishes its job tho whole program would end. In most cases this is not what we want to. When ROS spin is called node just waits until new calling of callback method [5].

2.4 Flexible Collision Library

Flexible Collision Library (FCL) [1] performs 3 types of proximity queries: collision detection, distance computation and tolerance verification. In this

project are collision checks queries preformed by this library because of its speed and easy integration with ROS.

Chapter 3

Virtual bumper

3.1 Collision map

Raw data from RealSense have to be converted into a collision map. Collision map is built by inserting points from point cloud in to an octree structure. In this step resolution of collision map is defined. Default value is set to 5 centimeters which means that the length of the smallest voxels at lowest octree level is 5 centimeters cube. This value has been chosen because the smallest obstacles which the robot normally encounters and needs to recognize, such as table leg, are on average this size. The collision map is represented as a set of occupied 5x5x5 centimeters voxels. On the figure below is example of collision map.

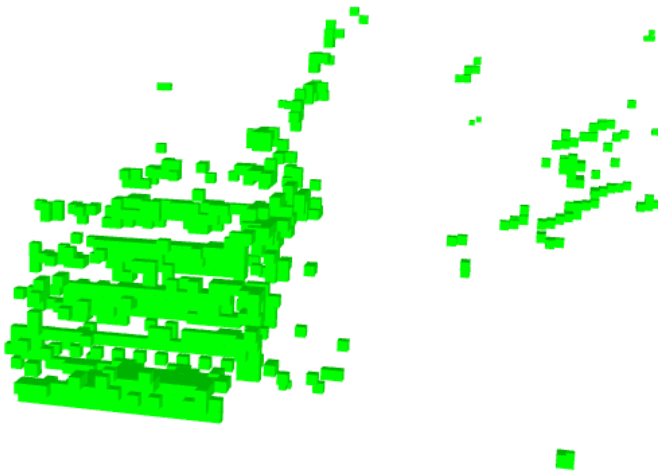


Figure 3.1: Collision map of stairs

3.2 Geometry

Depending on two flipper modes that are used most often. The standard observation mode when flippers are up which maximizes viewfield for the

sensors is used for traversing a flat terrain. In the approaching mode, the front flippers are tilted up in order to allow climbing up obstacles. In both cases, the robot pushes a virtual box which is 85 centimeters long, 60 centimeters wide and 50 centimeters high. That is the same size as the robot itself.

In observation flippers configuration, the box is in front of the robot and is lifted up as shown in figure 3.2. This lift is used because the robot can traverse small object even without the use of approaching state of flippers.

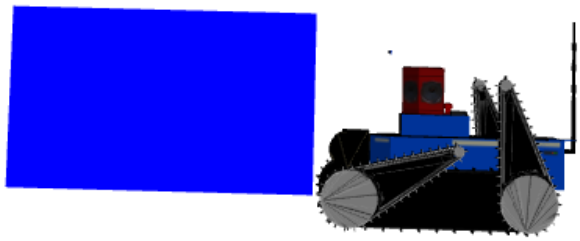


Figure 3.2: Bumper when flippers are up

In figure 3.3, the second state, which requires the box to be tilted up, is visualized. The angle between the ground and the box is 0.720 radians which is slightly less than flippers angle which is 0.786 radians. The angling of the box ensures that the robot with approaching flippers state can drive, for example, up on a stair without the bumper stopping it. Nevertheless, it still enables detection of obstacles which are too high to traverse. Proper detection of obstacle which are too high is the reason why angle of the box is smaller than the flippers angle, flippers have to get on the obstacle to traverse it therefor smaller angle of the bumper box cover the obstacles that would not fit under the flippers.

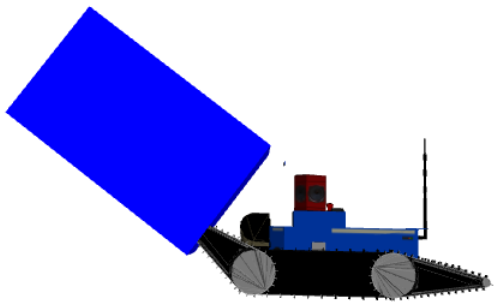


Figure 3.3: Bumper when flippers are in approaching state

3.3 Collision checking

The developed algorithm detects an overlap of a cube from the collision map with the virtual box, which represents the bumper. Each cube is individually checked and the number of cubes colliding with the bumper is summed into an output. If the output number exceeds the threshold, virtual bumper evaluates the situation as collision and stops the robot.

Setting up right threshold is essential for the Proper functioning of the bumper. In default setup where resolution of collision map is 5 centimeters, threshold is set on 25. This number have been calculated from tests which are described in chapter 6. Threshold can be set through parameter so its easy to adapt algorithm to specific environment. In flat indoor areas where are main obstacles walls can be threshold higher on the other hand in areas where is expected for example collisions with table foots chairs etc. threshold can be lowered.

Each point cloud data contains information about space in 50 centimeters distance and further. Hence, it is impossible to analyze the space in less than 50 centimetres distance from the robot without moving the sensor on the back of the robot, where can be difficult make vista forward because of other components of the robot. Therefor all collision check computations must be done before object gets in to the blind zone.

3.4 Speed of the algorithm

The RealSense camera can capture point clouds on 60 Hz frequency[12], which is immense amount of data. Based on robots maximum speed 0.3 meters per second we decided to set RealSense frame rate to 6 Hz, which gives us enough time to react without consuming too much computational resources.

Under normal conditions, it is necessary to analyze hundreds of thousands points from point clouds. A typical 3D point cloud is converted into a collision map with 800-3000 occupied voxels, which means that algorithm have to consequently undertake 12000 collision checks every second, nevertheless, significantly more could be required in environments with higher complexity.

The algorithm processing time can be decreased by lowering collision map resolution or by selecting some point clouds. However, these two methods lead to decreased reliability of the bumper. Ideally, both approaches could be combined to acquire the best optimization.

Still the method rate limiting step is conversion between the point cloud to octree structure, which takes 1000 times more time than collision checking.

Chapter 4

Implementation

Virtual bumper is standard ROS node with dependencies on those packages:

- geometry_msgs
- sensor_msgs
- tf
- tf2
- pcl_ros
- pcl_conversions

FCL library is used to perform collision check.

4.1 Parameters

Virtual bumper node have these parameters with predefined default values:

```
node.param("octMapResolution", octMapResolution, 0.05);
node.param("robotSizeX", robotSizeX, 0.85);
node.param("robotSizeY", robotSizeY, 0.6);
node.param("robotSizeZ", robotSizeZ, 0.5);
node.param("threshold", threshold, 20);
```

All length variables are in meters.

The length of an edge of each cube in collision map is defined in `octMapResolution` variable. Robot size can be changed by `robotSizeX`, `robotSizeY` and `robotSizeZ`. Changing of size parameter is useful when the robot is furnished with equipment that enlarges its range. When the threshold parameter is exceeded the robot stops. How to run program with parameters is wrote in next chapter in section Running 5.2.2.

4.2 Architecture

In constructor of virtual bumper publishers and subscribers are set. Important computations are made in methods:

- pointCloudCallback
- collisonDetect
- generateBoxesFromOcTree
- collisionHandler

Other methods ensures visualization and updating of global variables.

4.3 Callback method for pointcloud

The method `pointCloudCallBack` is called every time when the new point cloud is captured. First, point cloud is transformed from `camera_depth_optical_frame` to `base_link`. By this transformation is secured that geometry of the bumper will not change when position of the RealSense is changed.

Next the condition where `collisonDetect` method is called is evaluated. If collision is detected `collisionHandler` is called otherwise restriction for maximal speed is removed.

4.4 Collision detect

Main computations are made in `collisonDetect` method. The return value is logical operator which determinate if collision have been found or not.

4.4.1 Iteration through point cloud

Iteration through point cloud is done by pcl [13] library. Points are pushed to the `octomap` structure one by one as the cycle loop through them.

```
pcl::PointXYZ p;
       for (j = 0; j < local_pcl.size(); j++) {
           p = local_pcl.at(j);
           octomap::point3d point(p.x, p.y, p.z);
           octPointCloud.push_back(point);

       }
```

From `octomap` is than made `OcTree` by adding origin of the point cloud which must be corrected by the transformation, which been made in `pointCloudCallBack`.

4.4.2 Collision object

For collision check collision objects and FCL request must be created. The virtual bumper box is created from information about robots size. Translation and rotation depends on flippers position as is described in chapter 3.2. Sample code is shown below:

```
if (!(frontLeftFlipper > -2)) {
      transformBumper.setTranslation(
          Vec3f(stampedTransform.getOrigin().x() + 0.65,
          0, stampedTransform.getOrigin().z() - 0.15));
      rotation = Quaternion3f(0, 0, 0, 1);
  } else {
      transformBumper.setTranslation(
          Vec3f(stampedTransform.getOrigin().x() + 0.55,
          0, stampedTransform.getOrigin().z() + 0.1));
      rotation = Quaternion3f(0, -0.33, 0, 1);
  }
  Matrix3f rotationMatrix;
  rotation.toRotation(rotationMatrix);
  transformBumper.setRotation(rotationMatrix);
  fcl::CollisionObject
      collisionObjectBumper(bumperBox, transformBumper);
```

Again values are corrected by the transformation.

From `OcTree` is created collision map in `generateBoxesFromOcTree` method. In the loop is individually checked each box from collision map with bumper box. Result number is than checked with threshold. If threshold is exceed true is returned otherwise false is returned.

Optimization

Under normal circumstances condition will return true immediately after number of collisions exceeds the threshold. This stops creation of markers and number of collision stay on same value like threshold. For testing purposes and right visualization these lines have to be commented:

```
if(inCollision >= threshold){
    return true;
}
```

4.5 Collision handler

If collision is detected, collision handler is called. Everything what have to be done in danger situation should be placed here. Now collision handler sets maximum speed limit to 0.

4.6 Flipper state

Virtual bumper also subscribes `flippers_state` topic. Flippers direction in radians are stored in global variables. Translation and rotation of virtual bumper box depends on this variable. Same position of left and right flipper is expected.

```
if (!(frontLeftFlipper > -2)) { ...
```

4.7 Visualization

Visualization can be turn off and on by defining `markersVisible` statement. All parts of code responsible for visualization wont be built if `markersVisible` is not defined.

Markers representing collision map is visualized in same loop as collision check. If optimization condition is active visualization of collision map will not be complete, because loop is broke when threshold is reached.

Chapter 5

Users guide

5.1 Installation

Virtual bumper as standard ROS package is build by `catkin_make` command in root of working space or by newer version `catkin build`. Single node can be build by `catkin_make virtual_bumper`.

The virtual bumper node is part of the TRADR project where is placed in `tradr/tradr-loc-map-nav` set of packages. The repository is managed by CIIRC [11] gitlab.

5.2 Setting up on robot

RealSense camera is equipped on the robot by 3D printed holder facing forward as show in figure 5.1 . USB 3.0 standard is necessary because of large data flow.

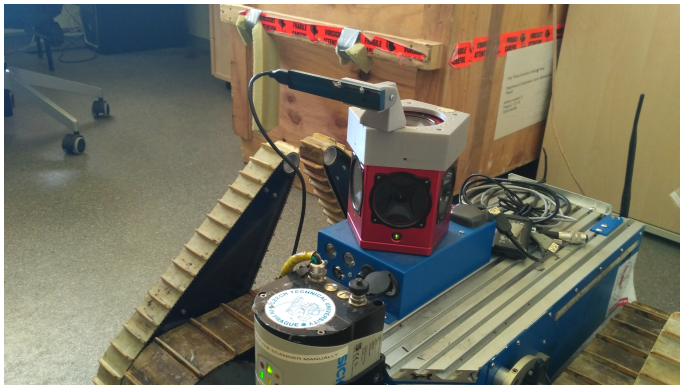


Figure 5.1: RealSense mount

5.2.1 Calibration

Camera and its model have to be synchronized by calibration in `robot.yaml` file. Programs `rtq_reconfigure` and `rviz` are used to achieve that. The robot is placed in front of the vertical object like wall or door, after that preview of point cloud in `rviz` have to be turn on. Furthermore `has_realsense`

check box must be checked in `rtq_reconfigure`. Next the point cloud is tilted to position where vertical object and the floor holds the right angle by changing value `realsense_tilt` in `rtq_reconfigure`. RealSense camera is normally deflected by 3 centimeters from middle of the robot because of holder structure. This shift is set in `realsense_shift_y` field. Example of calibration is shown in figure 5.2.

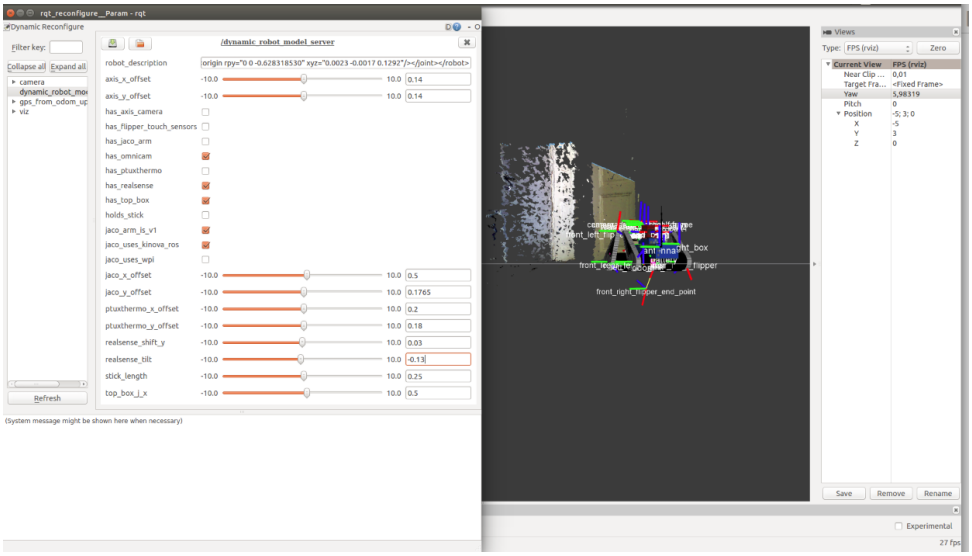


Figure 5.2: RealSense calibration

Accordingly information is written to `yaml` file which is opened by `rosed nifti_launchers robot.yaml` command. Pattern of file is demonstrated in figure 5.3.

```
  holds_stick: False
  stick_length: 0.25

  has_flipper_touch_sensors: False

  has_axis_camera: False
  axis_x_offset: 0.059
  axis_y_offset: 0.08

  # PTU Xtion Thermo Sensor
  has_ptuxthermo: False
  ptuxthermo_x_offset: 0.20
  ptuxthermo_y_offset: 0.18

  has_realsense: True
  realsense_tilt: 0.0
  realsense_shift_y: 0.03

viz:
  camera1394_driver:
    guid: ''
  lady_saw:
    camera_name_format: '_1232x1616_c%i'

omnicam_sensors_calib:
  transforms:
  - [0.042087, -0.001706, -0.000359, -0.500706, 0.499767, -0.499686, 0.499840, 'omnicam', 'camera_0']
  - [0.011469, -0.040128, -0.000085, -0.110544, 0.698795, -0.697954, 0.111021, 'omnicam', 'camera_1']
```

Figure 5.3: Pattern of robot.yaml file

5.2.2 Running

After calibration, node can be ran by writing:
`rosrun virtual_bumper virtual_bumper_node` in robots command line.
Example of starting the node with parameters is below.

```
rosrun virtual_bumper virtual_bumper_node _threshold:=20
_octMapResolution:=0.05 _robotSizeX:=0.85
_robotSizeY:=0.6 _robotSizeZ:=0.5
```

Chapter 6

Experiments

We designed 8 scenes where we made tests:

- Stairs
- Scene where is pallet on floor
- Too high obstacle
- Tilted plane
- Wall
- Small white pillar
- Small black pillar
- Big pillar

On this scenes 21 tests were ran. Model situations were recorded in to bag files and all sections were labeled as danger or safe. When the obstacle appears in front of the robot in distance shorter than robots length situation were labeled as danger when robots flippers are in observation position. With approaching state of flippers is danger distance considered as half of robots length. Number of detected collisions between collision map and bumper box in time were captured. From recorded data we set the threshold parameter to 25. However different environments requires different settings. Therefore this value is rather orientation and can be easily corrected if it is needed.

All model situations ends without collision, thus successful detection of danger state is when number of collisions exceeds threshold in danger part of the experiment. If number of collisions exceeds threshold during the safe part of an experiment false positive observation have been made.

Further in this chapter all danger data in graphs will be colored by red and safe data will be colored by green. The threshold value is displayed by the blue horizontal line.

6.1 Stairs

On the scene with stairs we ran 4 tests. Picture of the scene captured by robots camera is shown on figure below. In first test robot drives near stairs with observation state of flippers. On the graphs can be seen that part in front of the stairs is almost with no collisions and after robot reaches the critical distance collisions are rising very quickly.

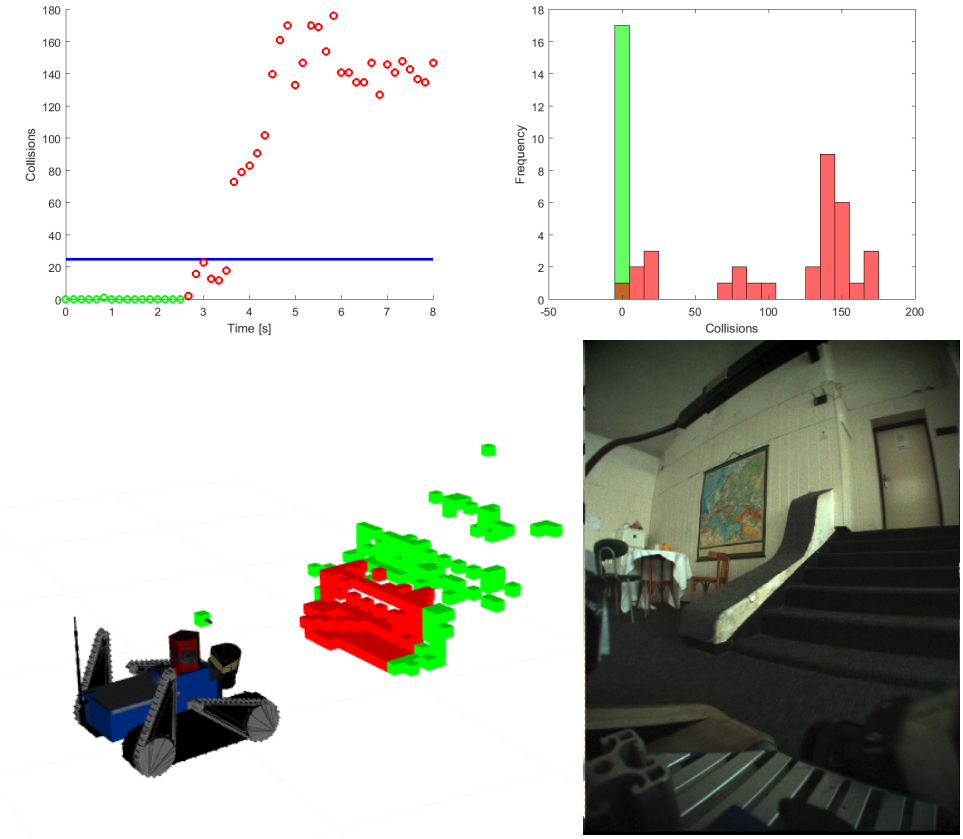


Figure 6.1: Drive to the stairs graph

Next two experiments expects no danger. The robot drives on stairs with proper position of flippers and than drives down on horizontal floor. In both experiments are values between 0 and 3, therefor safe situation were successfully detected. On collision map robot is facing the floor and no collision is detected.

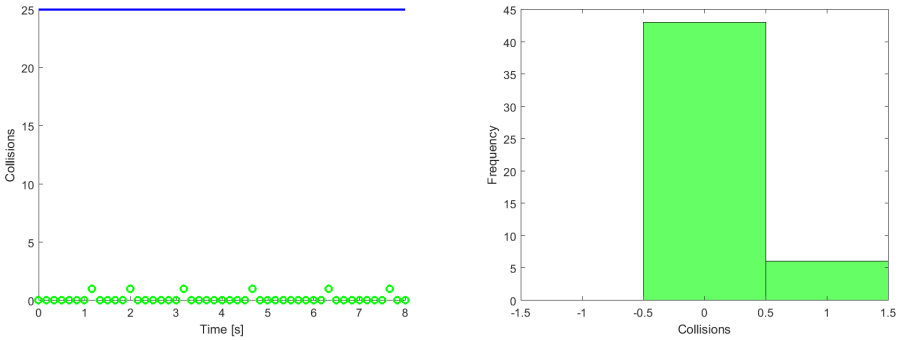
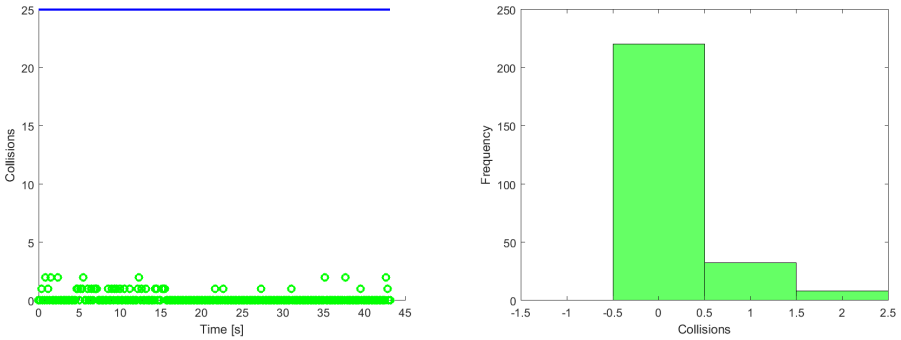


Figure 6.2: The robot approaching to stairs and drive on them



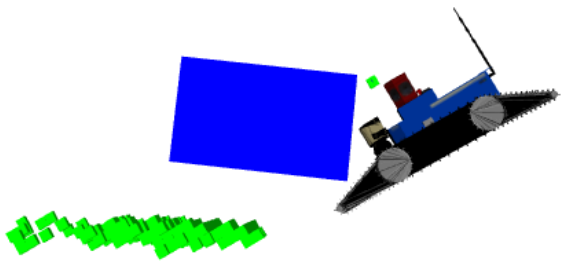


Figure 6.3: The robot drive down the stairs

The course of the last test shows how robot in front of the stairs change flippers state from observation to approaching. Thus he change danger state to safe and drive up the stairs. On figure below is shown collision map before and after flippers chage.

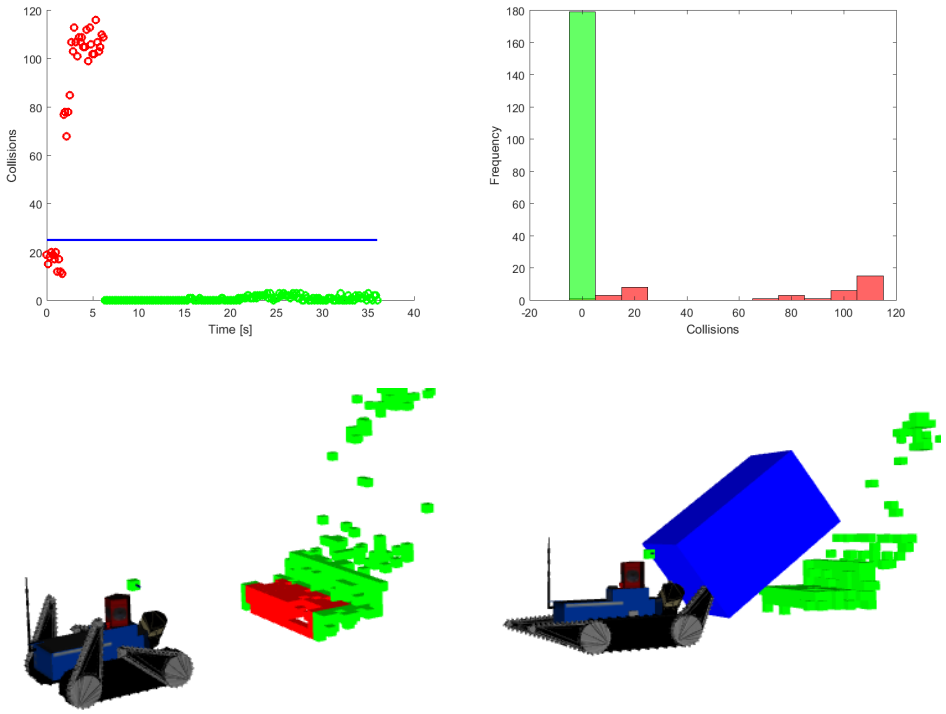


Figure 6.4: Change of the flippers state and drive on the stairs

6.2 Pallet

In pallet location were made 2 tests in which robot stops in front of the pallet and drive on the pallet. First test with flippers up comprise danger situation when the robot gets to close to pallet. The second one contains no danger due to approaching flippers state.

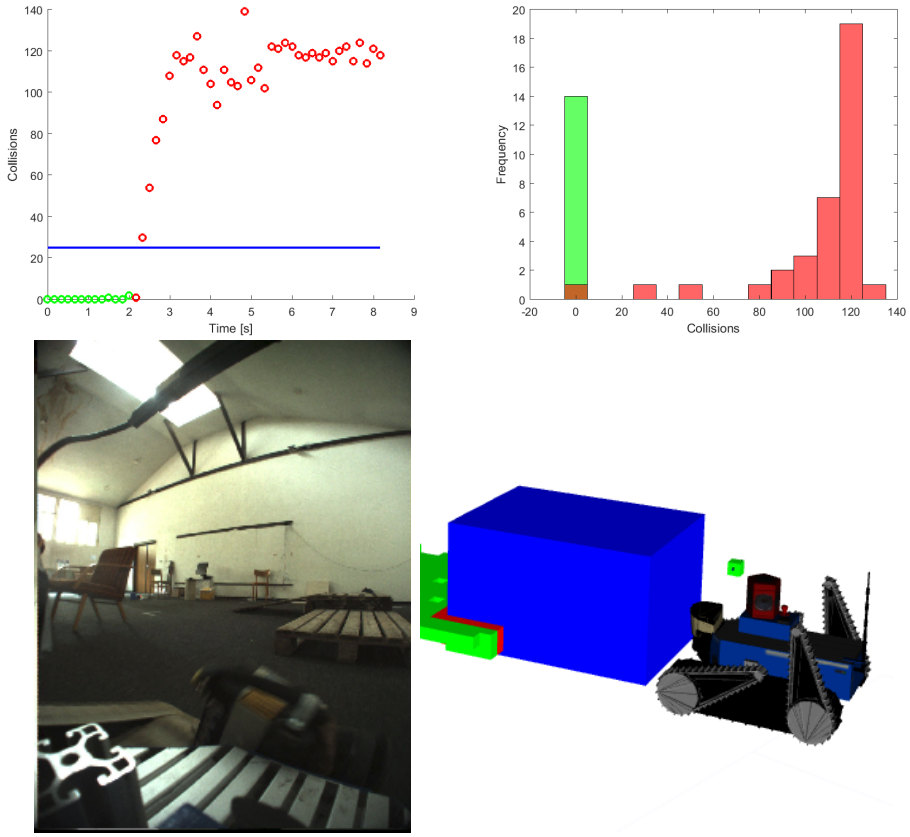


Figure 6.5: Drive towards the pallet

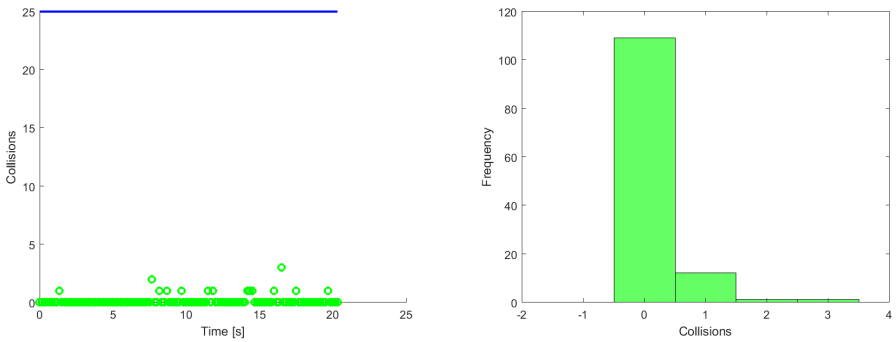


Figure 6.6: Drive on the pallet

6.3 High obstacle

Obstacle is defined too high if the robot can not traverse over it. Two pallets were placed one on each other as our high obstacle. The robot drives towards the pallets with the observation state of flippers and the approaching. In both situations high collision number is desirable. Nevertheless collisions with too high obstacle when flippers are in approaching settings is difficult to recognize because it has same shape like stairs, where the opposite detection is wanted. Thus the collision numbers in first test are significantly higher than in the second one.

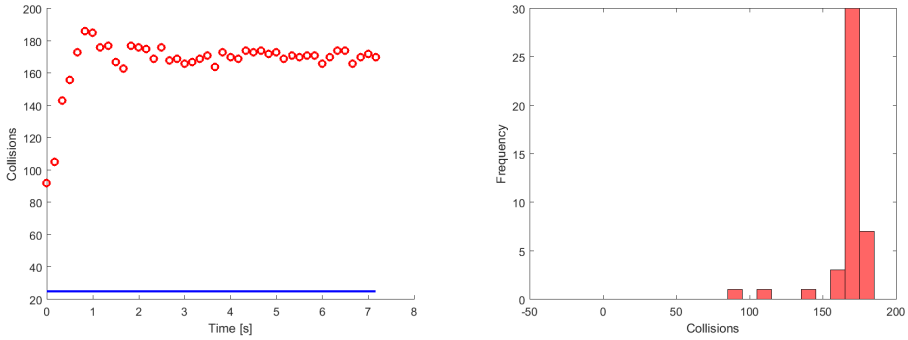


Figure 6.7: Too high obstacle, flippers observation

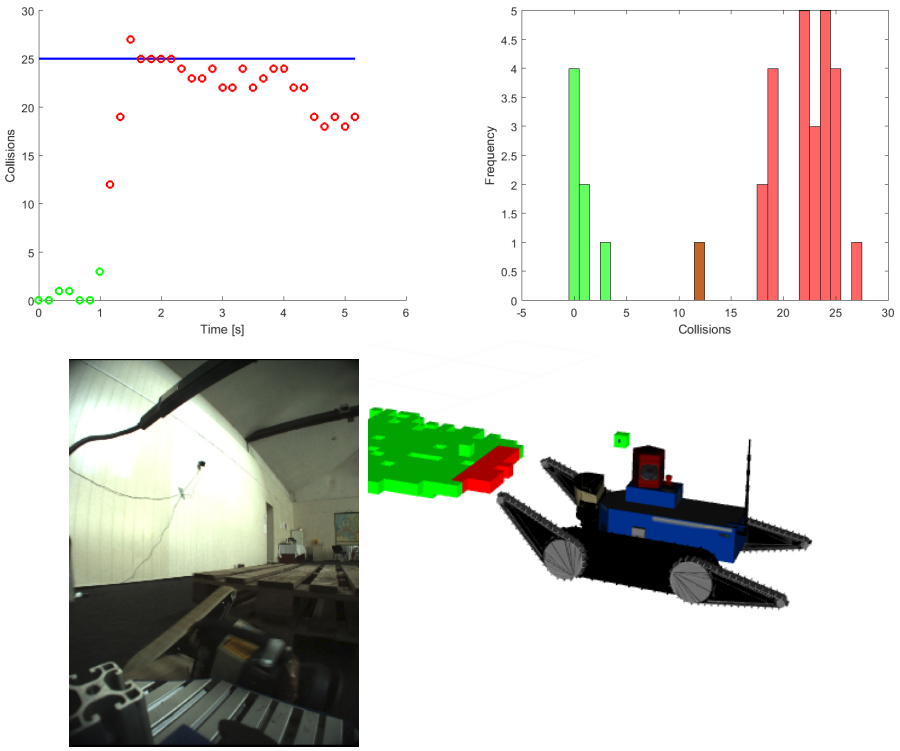


Figure 6.8: Too high obstacle, flippers approaching

6.4 Tilted plane

Drive on tilted plane is possible with both flippers configuration, observation and approaching. However because of the bumpers design with observation flippers state is tilted plane recognized as a danger obstacle as shown in graph 6.9. Solution of this problem is to change flippers state to approaching, which will eliminate detection of collision and it is also safer because the robot is Less prone to overturning. Data from drive on tilted plane with approaching flippers are show in figure 6.10.

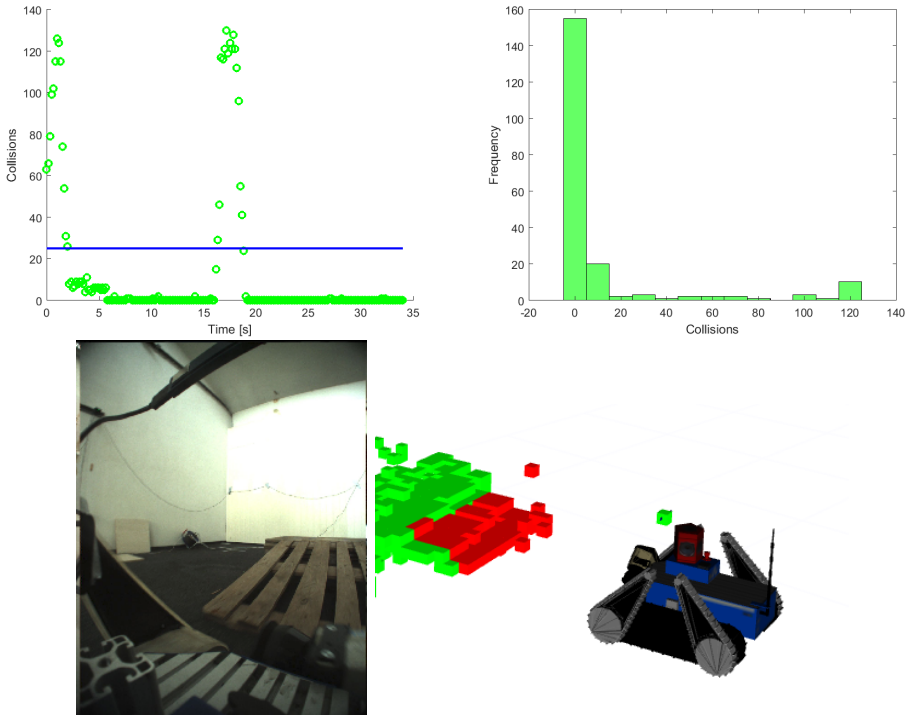


Figure 6.9: Tilted plane, flippers observation

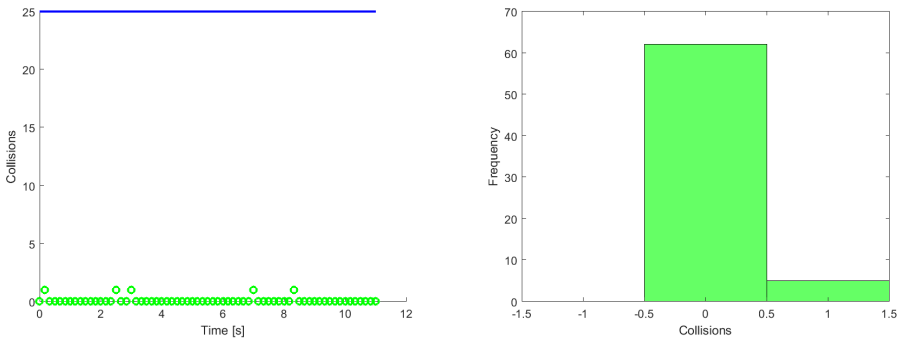


Figure 6.10: Tilted plane, flippers approaching

6.5 Wall

Wall is the same problem as the too high obstacle. Only in this case should be results more significant, but because overexposure of RealSense, point cloud did not captured the whole obstacle. Stereoscopic camera was not able to properly measure data on the monolithic white wall. Still enough collisions for evaluate danger situation were recognized.

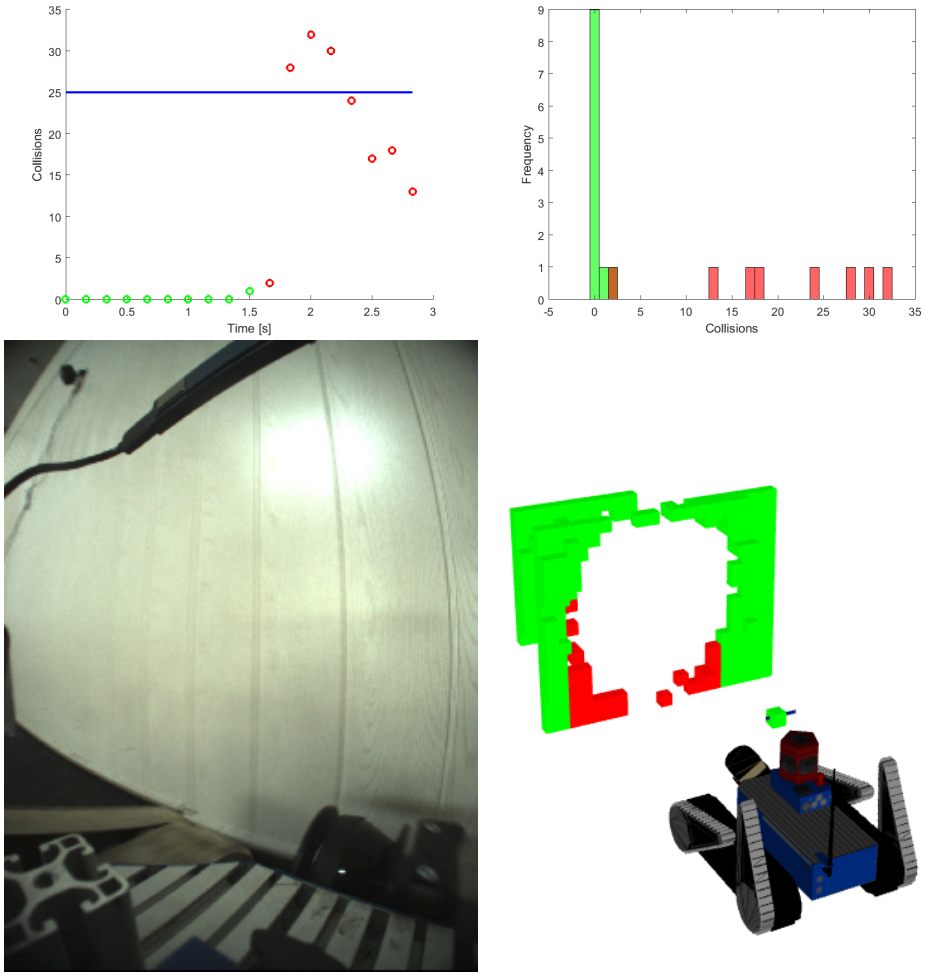


Figure 6.11: Robot approaching to the wall

In second test the robot moved towards the wall under an angle. In this situation RealSense captures the wall correctly, as shows figure below. This also corresponds with data in graphs where the collision numbers are moving around one hundred per pointcloud.

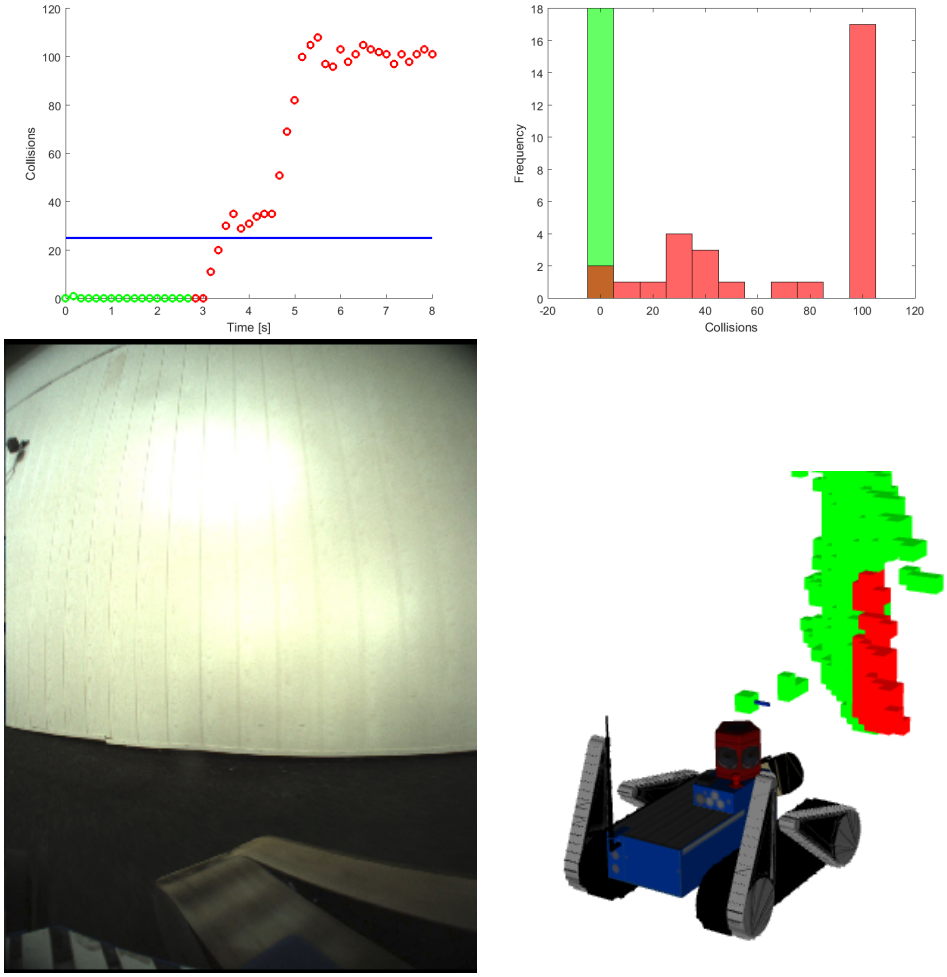


Figure 6.12: The wall under angle

6.6 Pillars

Our 3 testing pillars has diameter from 5 to 20 centimeters and different colours.

First white pillar with diameter 7 centimeters and second black with diameter 5 centimeters were tested in two ways. Approaching to the pillar, where center of the robot and the pillar were in one line was content of the first test. In second experiment was in one line the truck of the robot and the pillar.

Results below show, that white pillar in first test was not detected. RealSense camera was not able to capture whole pillar due to overexposure.

Black pillar was recognized, but the collision numbers were very close to threshold. In effort of better recognition of the thin pillars, resolution of collision map has to be increased or threshold decreased.

Large black pillar with diameter of 20 centimeters were tested on driving towards with the observation flippers state and the approaching. In both tests was successfully recognized as danger obstacle. however in second test was collision numbers again very close to threshold.

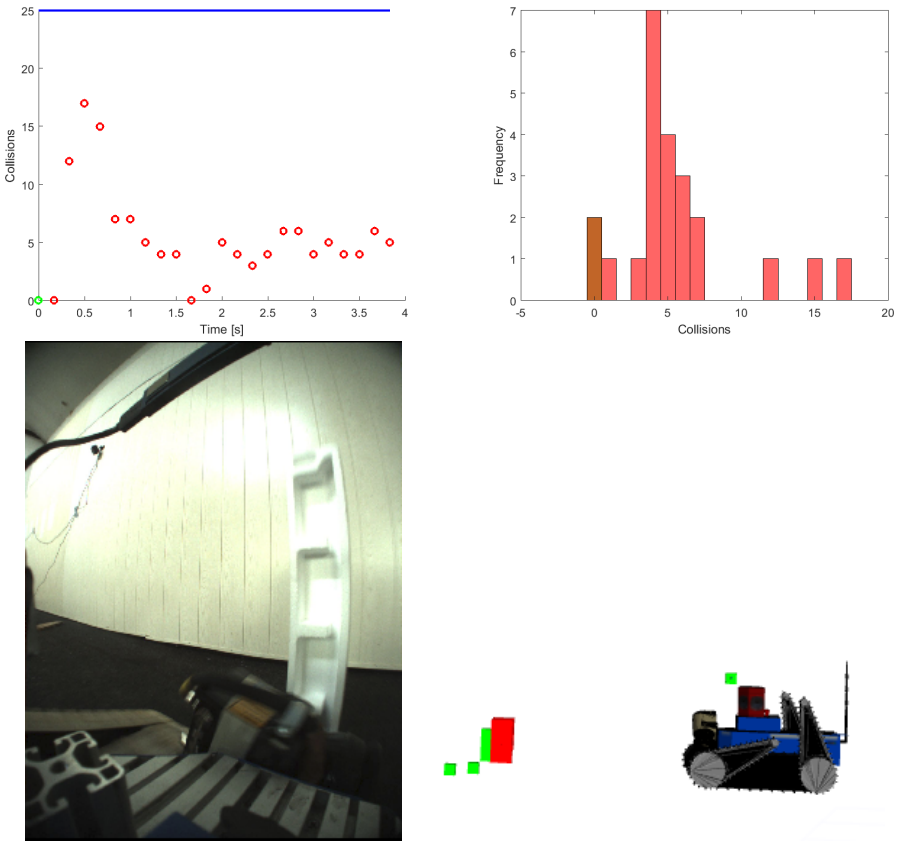


Figure 6.13: White pillar, approaching with middle of the robot

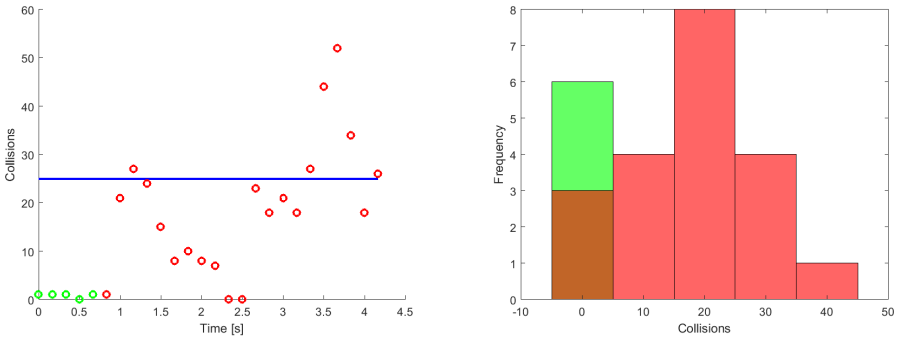


Figure 6.14: White pillar, approaching with the truck of the robot

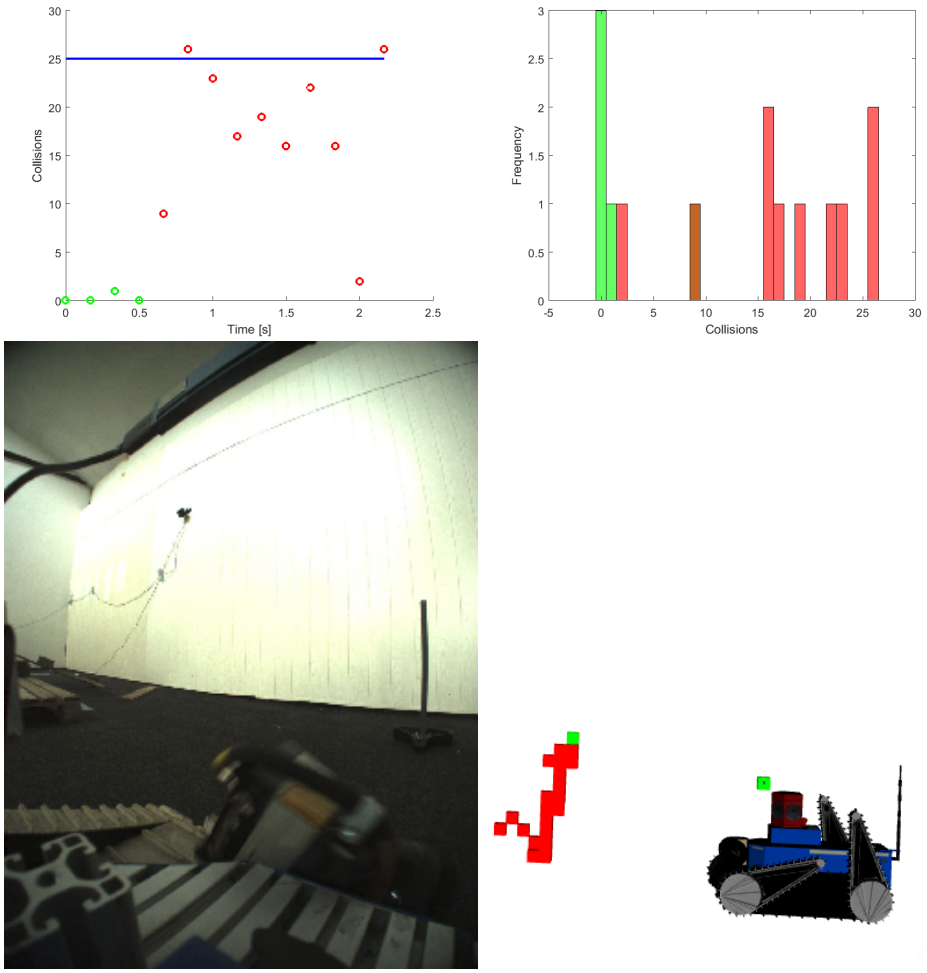


Figure 6.15: Black pillar, approaching with middle of the robot

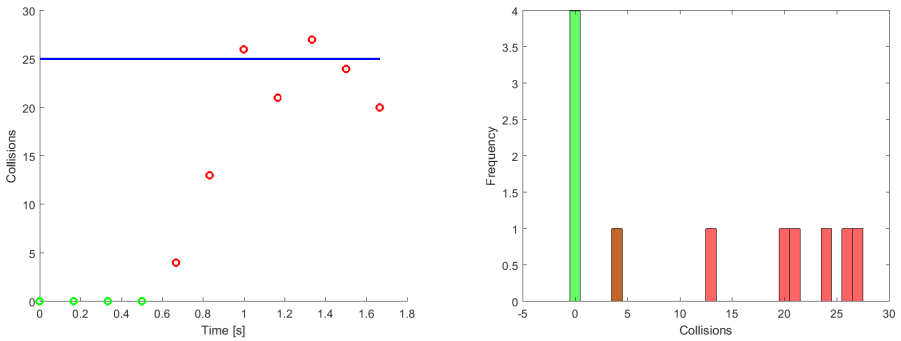


Figure 6.16: Black pillar, approaching with the truck of the robot

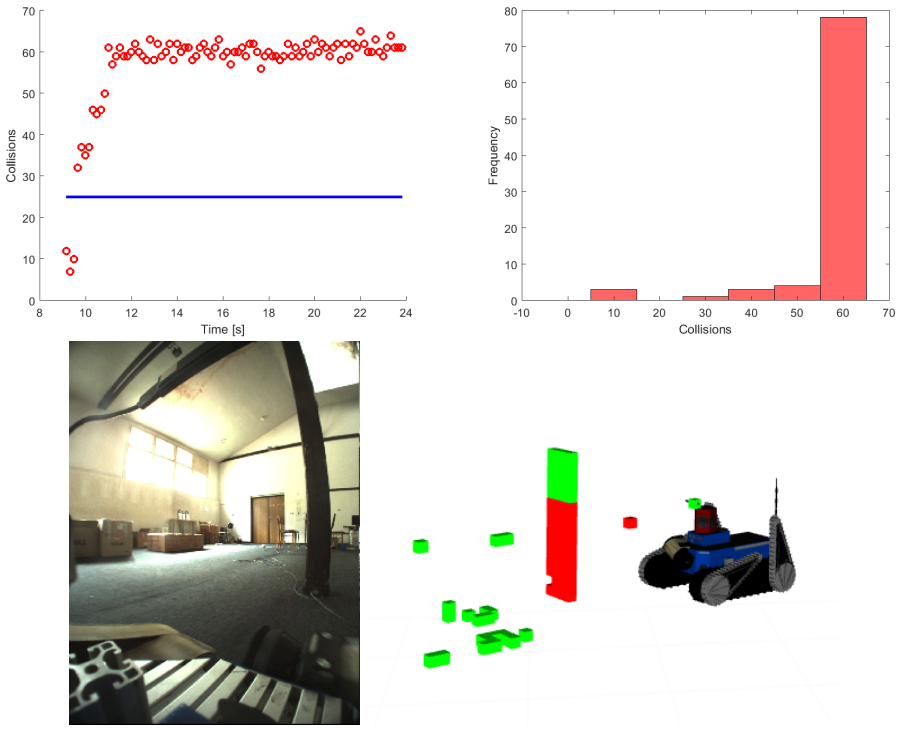


Figure 6.17: Large black pillar, flippers observation

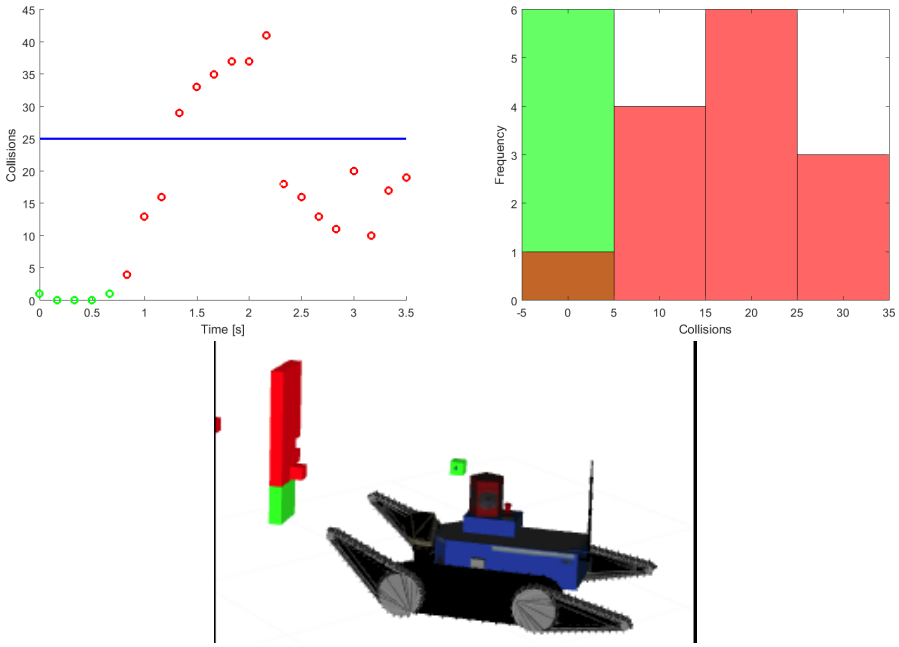


Figure 6.18: Large black pillar, flippers approaching

6.7 Testing drive

After learning the algorithm and setting the threshold on 25, testing drive with running node on the robot were made. Pillar test represented by foot of the table, stair tests and the wall tests were made. All situations were successfully recognized. In testing drive bumper were set to slow down the robot instead of stopping him, because function which allows manually overtake speed restriction is not implemented yet. Thus stopping the robot would leads to dead lock.

All bag files and videos can be downloaded here `http://ptak.felk.cvut.cz/tradr/visuals/bagfile_crawler/index.html`

6.7.1 Pillar test

In the figure below is shown process of slowing down in front of the table foot from the test drive. On first picture robot starts approaching towards table. Middle picture shows robot in full speed. On the left picture robot detects obstacle and slowed down.

Figure 6.19: Pillar test

6.7.2 Stairs tests

On following figures are demonstrated how robot drives on stairs and down the stairs. In both cases no collision was detected and robot easily drive through.

Figure 6.20: Test drive up the stairs

Figure 6.21: Test drive down the stairs

Chapter 7

Conclusion

We described how to design virtual bumper for the TRADR robot from depth data. ROS package in C++ which can stop the robot in front of the danger obstacles have been developed. We demonstrated functionality of this algorithm on several examples and data for further experiments were recorded during development. Described algorithm can be smoothly placed in any ROS project.

Simple calibration method has been described. However it is not automatized.

7.1 Limits of the method

Because RealSense sensor is facing forwards we can not detect holes. The geometric design of the bumper assumes that approaching flippers arrangement is used for climbing and observation for moving on flat surfaces. This assumption causes no problem on most of the scenes. However exceptions like driving on tilted plane with observation state of flippers described in chapter 6.4 or moving in place with low ceiling like tables with approaching state of flippers causes false alarms. In the case with low ceiling the volume of the bumper box is elevated and therefore colides with the ceiling like is show on figure 7.1 And all detection depends on point clouds which quality does not have to be always good.

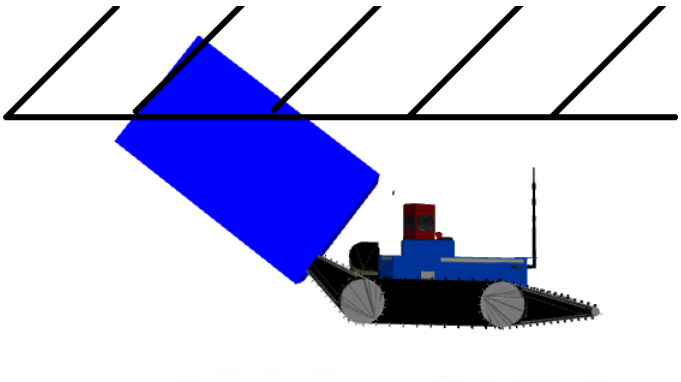


Figure 7.1: Bumper under low ceiling

Bibliography

[1] Flexible collision library `https://github.com/flexible-collision-library/fcl` visited: 14-5-2017

[2] Karel Zimmermann, Petr Zuzanek, Michal Reinstein, and Vaclav Hlavac *Adaptive Traversability of Unknown Complex Terrain with Obstacles for Mobile Robots* published Robotics and Automation (ICRA), 2014 IEEE International Conference on 31 May-7 June 2014

[3] Jia Pan, Ioan A. Şucan, Sachin Chitta, Dinesh Manocha *Real-time collision detection and distance computation on point cloud sensor data* Published in: Robotics and Automation (ICRA), 2013 IEEE International Conference on 6-10 May 2013

[4] About ROS `http://www.ros.org/about-ros/` visited: 4-5-2017

[5] ROS wiki `http://wiki.ros.org/` visited: 4-5-2017

[6] Long-Term Human-Robot Teaming for Disaster Response `http://www.tradr-project.eu/` visited: 11-5-2017

[7] Bullet `http://bulletphysics.org/` visited 14-5-2017

[8] Open dynamics engine `http://www.ode.org/` visited: 14-5-2017

[9] A Proximity Query Package `http://gamma.cs.unc.edu/SSV/` visited: 14-5-2017

[10] Thomas C. Hudson Ming C. Liny Jonathan Cohen Stefan Gottschalk Dinesh Manocha *V-COLLIDE: Accelerated Collision Detection for VRML* published in: Department of Computer Science University of North Carolina

[11] The Czech Institute of Informatics, Robotics and Cybernetics `https://www.ciirc.cvut.cz/` visited 19-5-2017

[12] Introducing the Intel® RealSense™ R200 Camera `https://software.intel.com/en-us/articles/realsense-r200-camera` visited: 21-5-2017

[13] Point cloud library`http://wiki.ros.org/pcl` visited: 21-5-2017

[14] J. Borenstein, Y. Koren *Obstacle avoidance with ultrasonic sensors* IEEE Journal on Robotics and Automation (Volume: 4, Issue: 2, Apr 1988)